Applying Ant Colony Optimization to Dynamic Binary-Encoded Problems

Michalis Mavrovouniotis and Shengxiang Yang

Centre for Computational Intelligence (CCI)
School of Computer Science and Informatics, De Montfort University
The Gateway, Leicester LE1 9BH, United Kingdom
{mmavrovouniotis, syang}@dmu.ac.uk

Abstract. Ant colony optimization (ACO) algorithms have proved to be able to adapt to dynamic optimization problems (DOPs) when stagnation behaviour is addressed. Usually, permutation-encoded DOPs, e.g., dynamic travelling salesman problems, are addressed using ACO algorithms whereas binary-encoded DOPs, e.g., dynamic knapsack problems, are tackled by evolutionary algorithms (EAs). This is because of the initial developments of the algorithms. In this paper, a binary version of ACO is introduced to address binary-encoded DOPs and compared with existing EAs. The experimental results show that ACO with an appropriate pheromone evaporation rate outperforms EAs in most dynamic test cases.

1 Introduction

Ant colony optimization (ACO) algorithms have shown good performance when applied to difficult optimization problems under static environments [4]. In general, ACO has been initially developed to tackle permutation-encoded problems [4]. There are also a few applications for binary-encoded problems, such as the multidimensional knapsack problem [1, 9–12]. Most of the existing ACO applications assume stationary environments. However, in many real-world applications we have to deal with dynamic environments, where the optimum changes and needs re-optimization.

Similarly to other nature-inspired algorithms [8, 16], ACO algorithms can adapt to dynamic changes since they are also inspired from nature, which is a continuous adaptation process. Practically, ACO can adapt to dynamic changes by transferring knowledge from past environments, using the pheromone trails, to speed up re-optimization [2]. The challenge to such algorithms when addressing dynamic optimization problems (DOPs) is that they suffer from the stagnation behaviour, where all ants construct the same solution from early stages of the algorithm execution. The adaptation capabilities of ACO rely on the pheromone evaporation where a constant amount of pheromone is deducted to eliminate pheromone trails that represent bad solutions that may bias ants to search to the non-promising areas of the search space. ACO algorithms have been successfully applied to dynamic extensions of the aforementioned permutation-encoded

problems, e.g., dynamic travelling salesman problems (TSPs) [14, 7] and dynamic vehicle routing problems (VRPs) [13, 15].

In this paper, we investigate the performance of ACO for solving dynamic binary-encoded optimization problems. Such problems have been successfully tackled by evolutionary algorithms (EAs) [19, 20], but not by ACO. Therefore, the original ACO framework is modified to construct binary-encoded solutions rather than permutation-encoded solutions (e.g., for the TSP) [5, 10]. In addition, we study the effect of introducing different pheromone evaporation rates and pheromone update policies into ACO for DOPs. As a result, a binary ACO framework, denoted ACO $_{\mathbb{B}}$, is established and integrated to the exclusive-or (XOR) DOP generator which can generate different dynamic test cases from given stationary binary-encoded problems [18]. Using this generator, an experimental study of comparing the proposed ACO$_{\mathbb{B}}$ with an existing genetic algorithm (GA) [19] and a population-based incremental learning (PBIL) algorithm [20].

The rest of the paper is organized as follows. Section 2 describes in detail the proposed ACO$_{\mathbb{B}}$. Section 3 describes the dynamic test environment for this study, including the details for the integration of ACO$_{\mathbb{B}}$ with the XOR DOP. Section 4 describes the experiments carried out on a series of different DOPs including relevant analysis. Finally, Section 5 concludes this paper with directions for future work.

2 Binary Ant Colony Optimization

The ACO metaheuristic consists of a population of μ ants that construct solutions and share information among each other via their pheromone trails. ACO was initially developed for the TSP [3] and later on applied for other optimization problems [4]. Only a few applications exist where the ants construct solutions with binary representation [5], mainly for the multidimensional knapsack problem [1, 9–12]. In most existing applications only stationary environments were considered.

The proposed binary-version of ACO (i.e., ACO$_{\mathbb{B}}$) closely follows the original ACO framework (see more details in [4]). Hence, in this section, we describe in detail only the ACO$_{\mathbb{B}}$. The main differences of the ACO$_{\mathbb{B}}$ from most existing binary-versions of ACO is that: it is designed to generally address binary-encoded problems rather than being dependent on the characteristics of the multidimensional knapsack problem, with the exception of the binary ant algorithm (BAA) [5]. This algorithm was also applied to dynamic problems but was not evaluated on the XOR DOP which is the most acceptable test suite for dynamic environments. The main difference of the proposed ACO $_{\mathbb{B}}$ from BAA relies on the way pheromone trails are updated.

2.1 Initialization

Typically, the pheromone table $\mathbf{P}$ of an ACO algorithm for the TSP, is defined as $\mathbf{P} = (\tau_{ij})_{n \times n}$, where n is the size of the problem instance (i.e., the number

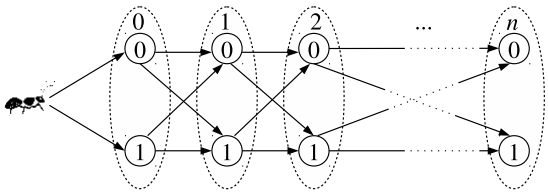

Fig. 1. Construction graph of ants for binary-encoded optimization problems of n size

of cities for the TSP) and τ_{ij} is the amount of pheromone between cities i to j. In contrast, the pheromone table of the binary framework of ACO (i.e., $\text{ACO}_{\mathbb{B}}$) is defined as $\mathbf{P} = (\tau_{ij})_{n\times 2}$. This is because the ants can only reach two possible values. The construction graph for optimization problems with a binary space is presented in Fig. 1, where the pheromone trails are associated to the corresponding directed arcs.

Let (i, j) denote the associated pheromone trail of state i (i.e., $i = 0, \ldots, n$) to value j (i.e., $j \in \{0, 1\}$). Initially, all the pheromone trails are set with an equal amount $\tau_0 \leftarrow \mu C^*, \forall (i, j)$ where C^* and μ are the optimal solution (or approximation) and the population size, respectively.

2.2 Constructing Solutions

Each ant k performs n construction steps and uses a probabilistic rule to select the next state to visit, where each state is associated with only two possible values as presented in Fig. 1. More precisely, the probability an ant k uses to select a value j for state i is defined as follows:

$$p_{ij}^k = \frac{\tau_{ij}}{\sum_{l\in\{0,1\}} \tau_{il}}, j \in \{0, 1\}, \tag{1}$$

where τ_{ij} is the associated pheromone trail of state i to value j. Typically, heuristic information is considered together with the existing pheromone trails on the conventional ACO. For example, when ACO is applied on the TSP, the inverse of the distance between cities is used as the heuristic information [3]. For the problems tackled in this paper, there is no available heuristic information and, thus, only the pheromone information is considered. In fact, the construction of solutions based on probabilities (e.g., the pheromone trails table) is similar with the probability vector used in PBIL [20]. However, the "probability vector" in $\text{ACO}_{\mathbb{B}}$ is maintained differently from the PBIL.

2.3 Pheromone Update Policy

The pheromone trails in $\text{ACO}_{\mathbb{B}}$ are updated by applying evaporation as follows:

$$\tau_{ij} \leftarrow (1 - \rho)\, \tau_{ij}, \forall (i, j), \tag{2}$$

where $\rho \in (0, 1]$ is the evaporation rate and τ_{ij} is the existing pheromone value. After evaporation, the best-so-far ant deposits pheromone as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{bs}, \forall(i, j) \in \boldsymbol{x^{bs}}, \tag{3}$$

where $\Delta\tau_{ij}^{bs} = C^{bs}$ and $\boldsymbol{x^{bs}} \in \{0, 1\}^n$ are the amount of pheromone that the best ant deposits and the best ant's solution, respectively, and C^{bs} is the fitness of the best-so-far ant. In contrast, the existing BAA algorithm allows all ants to deposit pheromone [5]. Since only the best ant deposits pheromone, the algorithm will quickly converge towards the best solution of the first iteration. Therefore, pheromone trail limits are imposed in order to avoid this behaviour such that $\tau_{min} \leq \tau_{ij} \leq \tau_{max}, \forall(i, j)$, where τ_{min} and τ_{max} are the lower and upper pheromone trail values [9, 17]. The upper pheromone trail value is modified whenever a new best solution is found such that $\tau_{max} = \mu C^{bs}$ (where initially $\tau_{max} = \tau_0$). The lower pheromone trail value is set to $\tau_{min} = \tau_{max}/2\mu$. In this way, the lower pheromone trail value is also modified whenever a new best solution is found.

2.4 Response to Dynamic Change

ACO algorithms are able to use knowledge from previous environments using the pheromone trails generated in the previous iterations. For example, when the changing environments are similar, the pheromone trails of the previous environment may provide knowledge to speed up the optimization process to the new environment. However, the algorithm needs to be flexible enough to accept the knowledge transferred from the pheromone trails, or eliminate the pheromone trails, in order to adapt well to the new environment.

ACO algorithms can be applied directly to DOPs without any modifications due to the pheromone evaporation. Lowering the pheromone values enables the algorithm to forget bad decisions made in previous iterations. When a dynamic change occurs, evaporation eliminates the pheromone trails of the previous environment from areas that are not visited frequently and may bias ants not to adapt well to the new environment. The adaptation via pheromone evaporation may be a sufficient choice when the changing environments are similar; otherwise, the pheromone trails may misguide ants towards non-promising areas in the search space.

3 Dynamic Test Environments

3.1 Generating Dynamic Environments

The XOR DOP generator [18, 20] can construct dynamic environments from any binary-encoded stationary function $f(\boldsymbol{x})(\boldsymbol{x} \in \{0, 1\}^n)$ by a bitwise XOR operator. Suppose the environment changes in every f algorithmic generations, the dynamics can be formulated as follows:

$$f(\boldsymbol{x}, t) = f(\boldsymbol{x} \oplus \boldsymbol{M}(k)), \tag{4}$$

where "$\oplus$" is the XOR operator (i.e., $1 \oplus 1 = 0, 1 \oplus 0 = 1, 0 \oplus 0 = 0$), $k = \lceil t/f \rceil$ is the index of the period and $\boldsymbol{M}(k)$ is the XORing mask that occurs incrementally and is defined as follows:

$$\boldsymbol{M}(k) = \boldsymbol{M}(k-1) \oplus \boldsymbol{T}(k), \tag{5}$$

where $\boldsymbol{T}(k)$ is an intermediate binary template randomly created with $m \times n$ ones. Parameters $m \in [0, 1]$ and f control the magnitude and frequency of change of a DOP, respectively. A higher value of m means severer dynamic changes, whereas a lower value of f means faster dynamic changes.

In this paper, four 100-bit binary-encoded problems are selected as the stationary problems to generate DOPs. Each problem consists of 25 copies of 4-bit building blocks and has an optimum of 100. The first one is the *OneMax* function, which aims to maximize the number of ones in a chromosome. The second one is the *Plateau* function, where each building block contributes four (or two) to the total fitness if its unitation (i.e., the number of ones inside the building block) is four (or three); otherwise, it contributes zero. The third one is the *RoyalRoad* function where each building block contributes four to the total fitness if its unitation is four; otherwise, it contributes zero. The fourth one is the *Deceptive* function, where the building block is a fully deceptive sub-function. Generally, the difficulty of the four functions for optimization algorithms is increasing in the order from OneMax to Plateau to RoyalRoad to Deceptive.

3.2 Integration of ACO with the XOR DOP Generator

EAs are typically integrated with the XOR DOP generator [19]. The binary template is applied to each individual within the population of an EA. In this way, the XOR DOP generator shifts the population to a different area in the fitness landscape. Similarly, the binary template can be applied to the solutions constructed by the ants within the $\text{ACO}_{\mathbb{B}}$.

Since the population in ACO is re-constructed on every iteration, the binary template is applied before the population is cleared for the new iteration. Hence, the fitness of the solutions constructed using the same pheromone trails before and after a dynamic change (whenever the binary template is applied) will differ depending on the magnitude of change.

4 Experimental Study

4.1 Experimental Setup

For each algorithm on a DOP, $R = 30$ independent runs were executed on the same environmental changes. The algorithms were executed for $E = 1000$ iterations and the overall offline performance is calculated as follows:

$$\bar{P}_{offline} = \frac{1}{E} \sum_{i=1}^{E} \left(\frac{1}{R} \sum_{j=1}^{R} P_{ij}^{*} \right), \tag{6}$$

where P^*_{ij} defines the fitness of the best-so-far ant since the last dynamic change of iteration i of run j [8].

Dynamic test environments are generated from the four aforementioned binary-encoded function, described in Section 3, using the XOR DOP generator with f set to 10 and 50, indicating quickly and slowly changing environments, respectively, and m set to 0.1, 0.2, 0.5, 0.8 and 1.0, indicating slightly, to medium, to severely changing environments, respectively. As a result, ten dynamic environments (i.e., 2 values of $f \times 5$ values of m) from each stationary function are generated to systematically analyze the algorithms on the DOPs.

4.2 Analysis of the Pheromone Evaporation Rate

To investigate the effect of the pheromone evaporation in $\text{ACO}_{\mathbb{B}}$, different evaporation rates, i.e., $\rho \in \{0.0, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, are selected. In Fig. 2, the offline performance of $\text{ACO}_{\mathbb{B}}$ with the different evaporation rates is presented and the following observations can be drawn.

First, the performance of $\text{ACO}_{\mathbb{B}}$ is degraded when $\rho = 0.0$ (i.e., pheromone evaporation is not used) in comparison with the performance of $\text{ACO}_{\mathbb{B}}$ when $\rho > 0.0$ (i.e., pheromone evaporation is used) in most DOPs. This is natural because the evaporation rate is the adaptation mechanism within $\text{ACO}_{\mathbb{B}}$. More precisely, pheromone trails distributed to the optimum of the previous environment are eliminated by pheromone evaporation to help ants generate new pheromone trails for the optimum of the current environment.

Second, when the evaporation rate is set to $\rho \in [0.1, 0.4]$, the performance of $\text{ACO}_{\mathbb{B}}$ is usually superior than when it is set to other values. This is because a higher value of ρ may destroy the knowledge gained from previous iterations, whereas a lower value of ρ may not help the adaptation process on the new environment. Usually, when the magnitude of change increases; a higher evaporation rate performs better. For example, on *OneMax*, *Plateau* and *RoyalRoad* with $f = 50$, an evaporation rate of 0.1, 0.2, 0.3 and 0.4 performs the best when the magnitude of change is 0.1, 0.2 and 0.5, 0.8 and 1.0, respectively.

4.3 Analysis of the Pheromone Update Policy

In this section, we investigate four pheromone update policies for $\text{ACO}_{\mathbb{B}}$ (with $\rho = 0.2$): 1) only the best ant deposits pheromone; 2) all ants deposit pheromone[1]; 3) only the best ant deposits pheromone and limits are imposed[2]; and 4) all the ants deposit pheromone and limits are imposed. In Fig. 3, the offline performance for the different pheromone update policies are presented and several observations can be drawn.

First, the performance is degraded when only the best ant deposits pheromone in most DOPs. This is because high concentration of pheromone trails are quickly

[1] Similar update policy with the existing BAA [5].

[2] This pheromone update policy is the one described in Section 2 and finally associated with $\text{ACO}_{\mathbb{B}}$.

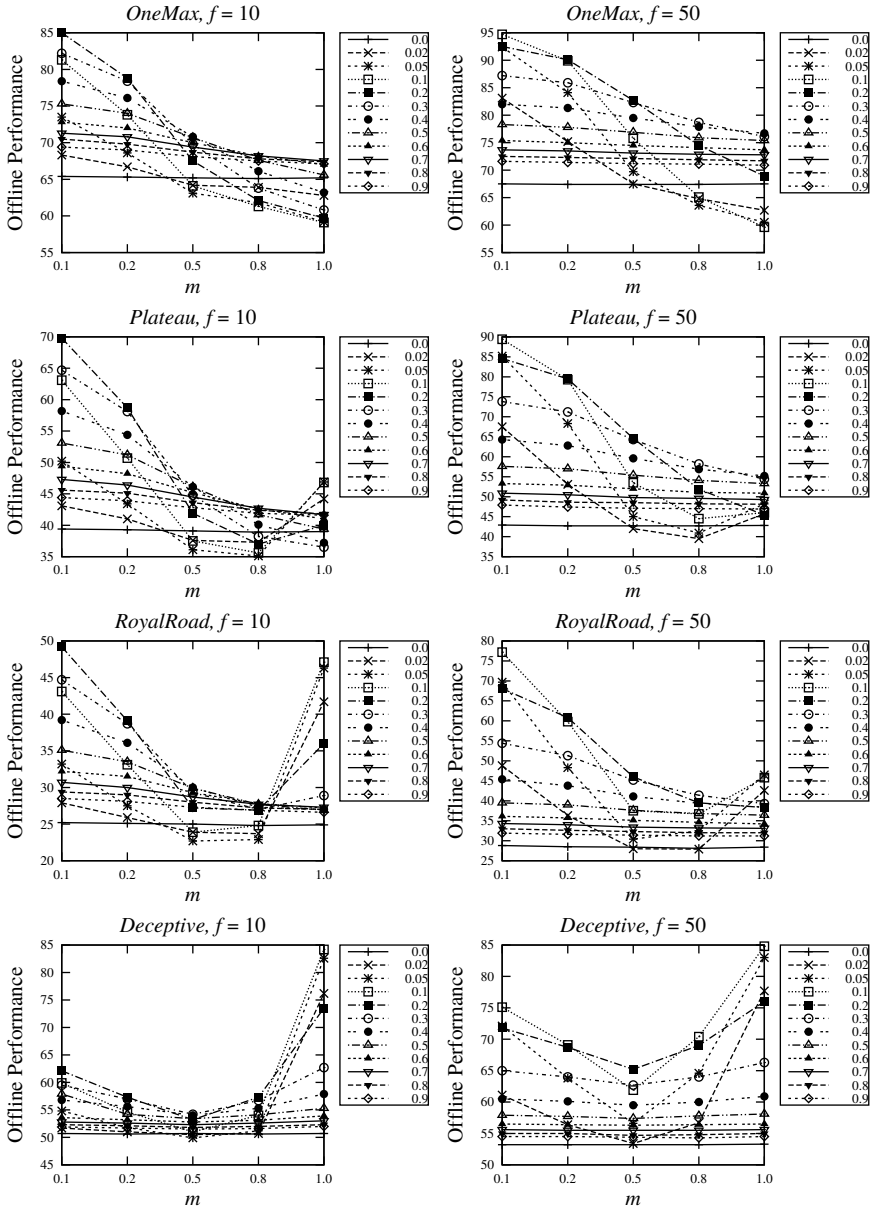

Fig. 2. Offline performance of $\text{ACO}_{\mathbb{B}}$ with different pheromone evaporation rates for different DOPs

generated to the solution of the best ant. Hence, stagnation behaviour occurs and the algorithm cannot adapt well to the changes. In contrast, the perfor-

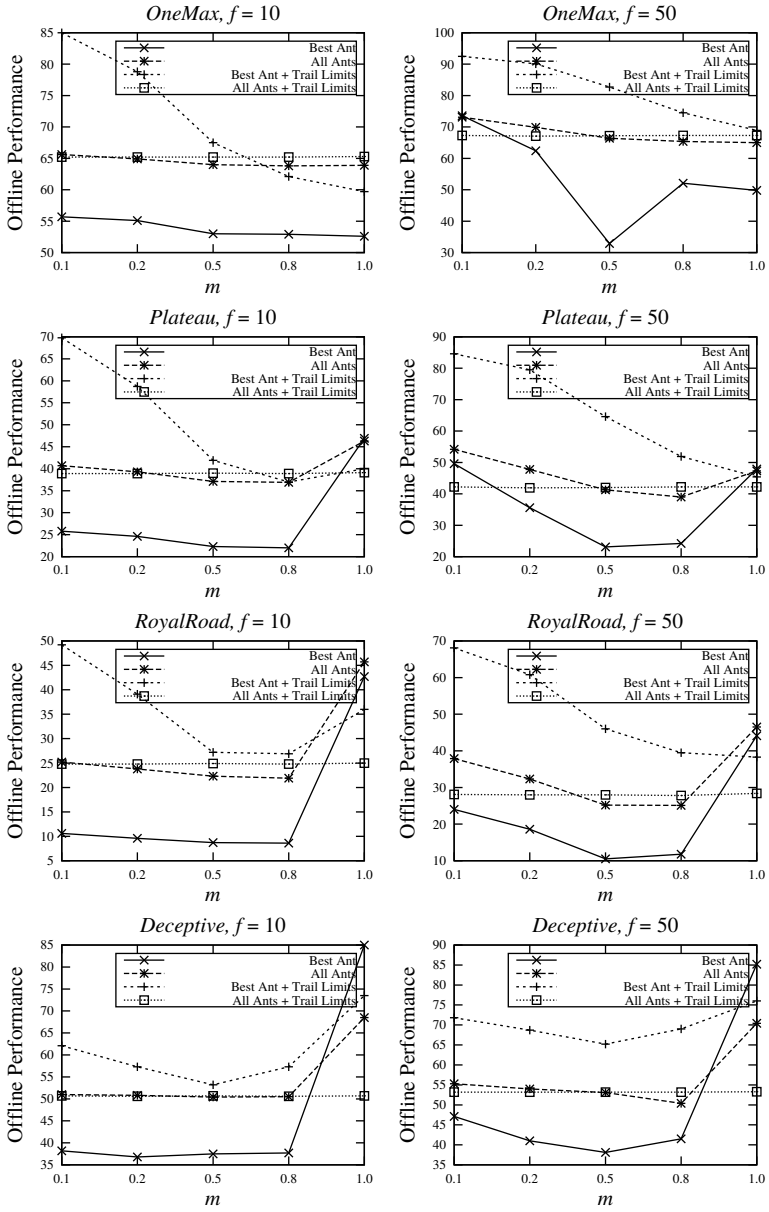

Fig. 3. Offline performance of $\text{ACO}_{\mathbb{B}}$ with different pheromone update policies for different DOPs

mance is better when all ants deposit pheromone because the pheromone trails are distributed among several solutions rather than to one solution.

Second, the performance is improved when trails limits are imposed when only the best ant deposits pheromone in most DOPs. This is because the stagnation behaviour described previously is addressed. The difference between the maximum and minimum pheromone trails is not significantly different, and thus, more chances are given to the less attractive areas (in terms of pheromone) to be explored. In contrast, the performance is similar when trail limits are imposed when all the ants deposit pheromone.

Third, when trails limits are not imposed in general, the performance of $\text{ACO}_{\mathbb{B}}$ is often improved when $m = 1.0$. When a DOP changes with $m = 1.0$ using the XOP DOP generator, it basically switches between two environments consecutively. Hence, memory enhanced algorithms may be more suitable in such special cases since the environments re-appear [19]. In fact, the pheromone table within $\text{ACO}_{\mathbb{B}}$ can be considered as an adaptive memory scheme. However, the pheromone trails cannot store exactly the solutions when trail limits are imposed.

4.4 Analysis of Algorithm Comparisons

In the experiments, we compare the proposed $\text{ACO}_{\mathbb{B}}$ with GA [19] and PBIL [20] used in DOPs. Since $\text{ACO}_{\mathbb{B}}$ is not enhanced with additional components to address DOPs the only the standard versions of GA and PBIL algorithms are used. The population size for all algorithms was set to $\mu = 120$ for a fair comparison. For GA the parameters were set to typical values as follows: generational, uniform crossover with $p_c = 0.6$, flip mutation with $p_m = 0.01$, and fitness proportionate selection with elitism of size 1. For PBIL the parameters were also set to typical values as follows: the learning rate $\alpha = 0.25$, mutation probability $p_m = 0.02$, mutation shift $\delta_m = 0.05$ and elitism of size 1. For $\text{ACO}_{\mathbb{B}}$, the parameters were set as follows: the evaporation rate $\rho = 0.2$.

The offline performance of $\text{ACO}_{\mathbb{B}}$ compared with the other algorithms is presented in Table 1. Kruskal–Wallis tests were applied followed by posthoc paired comparisons using Mann–Whitney tests with the Bonferroni correction. Moreover, the dynamic behaviour of the algorithms on different DOPs is presented in Fig. 4. From the experimental results, several observations can be made by comparing the behaviour of the algorithms.

First, $\text{ACO}_{\mathbb{B}}$ outperforms both PBIL and GA in most DOPs with $m = 0.1$, $m = 0.2$, $m = 0.5$ and $m = 0.8$. It can be clearly observed from Fig. 4 that $\text{ACO}_{\mathbb{B}}$ maintains higher fitness than the competing algorithms during almost all the environmental changes. The mutation operator applied directly to the evolving population of GA may help the population to adapt to dynamic changes but slows down the optimization process. PBIL maintains the highest fitness on the initial environment (between iterations $0 - 100$) but then it is unable to maintain it. In contrast to the GA's case, the mutation operator applied to the probabilistic vector of PBIL may not be sufficient to move the population from the previously converged optimum (except when $f = 50$ with $m = 0.1$ and $m = 0.25$).

Table 1. Experimental results of the algorithms regarding the offline performance. Bold value or values indicate(s) that the algorithms are significantly better or insignificantly different than the other algorithms, respectively

	$f = 10$					$f = 50$				
DOPs, $m \Rightarrow$	0.1	0.2	0.5	0.8	1.0	0.1	0.2	0.5	0.8	1.0
Algorithms	*OneMax*									
GA	73.4	69.6	64.4	**62.8**	**62.0**	82.7	79.2	72.3	68.0	65.6
PBIL	74.8	66.8	57.7	55.1	54.2	**92.6**	86.0	69.6	60.0	55.6
$ACO_{\mathbb{B}}$	**85.0**	**78.8**	**67.5**	62.1	59.7	**92.5**	**90.1**	**82.7**	**74.5**	**68.9**
Algorithms	*Plateau*									
GA	57.6	49.9	39.5	36.8	41.3	75.6	69.0	56.4	49.0	45.8
PBIL	50.5	39.6	29.1	29.1	**49.4**	**84.0**	69.5	43.9	40.3	**48.0**
$ACO_{\mathbb{B}}$	**69.8**	**58.7**	**41.9**	**37.0**	40.0	**84.6**	**79.5**	**64.6**	**51.9**	45.4
Algorithms	*RoyalRoad*									
GA	44.1	36.3	**27.1**	**27.1**	39.8	66.6	57.9	44.8	**40.3**	41.6
PBIL	25.1	19.7	14.5	15.5	**48.3**	56.5	39.6	23.6	25.9	**45.2**
$ACO_{\mathbb{B}}$	**49.2**	**39.1**	**27.2**	26.9	36.0	**68.1**	**60.8**	**46.0**	39.5	38.3
Algorithms	*Deceptive*									
GA	55.1	52.9	51.2	52.8	67.1	64.2	61.5	58.7	62.2	72.5
PBIL	55.0	49.4	44.9	47.5	**86.8**	69.7	64.3	57.0	65.5	**86.9**
$ACO_{\mathbb{B}}$	**62.1**	**57.5**	**53.2**	**57.3**	73.5	**71.8**	**68.7**	**65.2**	**69.0**	76.0

Second, PBIL outperforms both GA and $ACO_{\mathbb{B}}$ in most DOPs with $m = 1.0$. We have previously described the cyclic characteristic of this specific dynamic case and that the pheromone table of $ACO_{\mathbb{B}}$ can be viewed as an adaptive memory scheme. Similarly, the probabilistic vector in PBIL can be also viewed as an adaptive memory scheme. Hence, the probabilistic vector of PBIL may be able to store and maintain information for the two environments that can be reused when they re-appear. In contrast, the information may be destroyed from the pheromone table of $ACO_{\mathbb{B}}$ by the pheromone evaporation.

5 Conclusions

In this paper, the application of ACO to binary-encoded optimization problems in dynamic environments is investigated. The proposed $ACO_{\mathbb{B}}$ is designed to construct binary-encoded solution biased by pheromone trails. The effect of using different pheromone update policies and pheromone evaporation rates is studied for $ACO_{\mathbb{B}}$ in DOPs.

The $ACO_{\mathbb{B}}$ is integrated with the XOR DOP generator and a series of dynamic test cases are systematically constructed from several benchmark stationary problems. From the experiments, several concluding remarks can be drawn. First, pheromone evaporation enhances the adaptation capabilities of $ACO_{\mathbb{B}}$. An evaporation rate between $\rho \in [0.1, 0.4]$ achieves the best performance in DOPs but it is dependent on the magnitude of change of a DOP: the higher the magnitude the higher the rate. Second, the use of pheromone trail limits address the

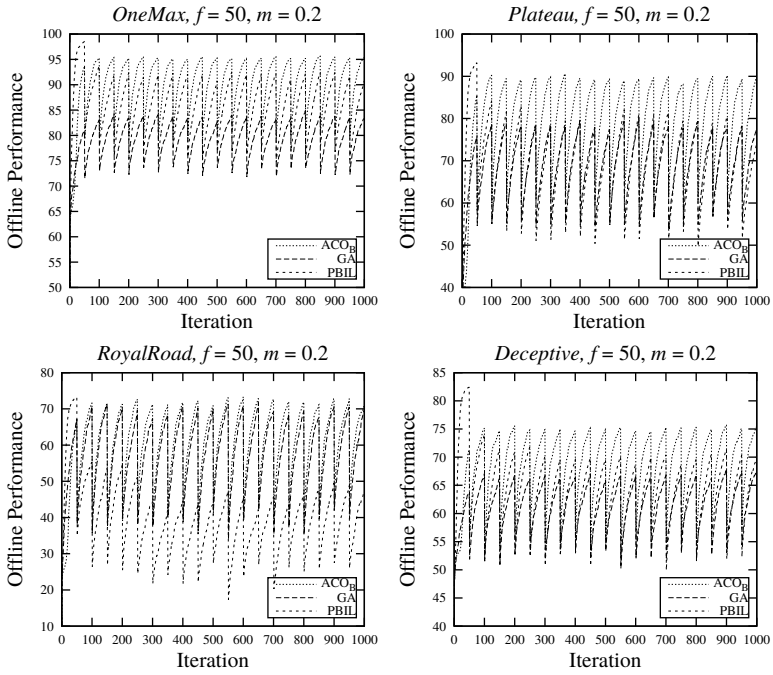

Fig. 4. Dynamic offline performance of algorithms on DOPs with $f = 50$ and $m = 0.2$

stagnation behaviour of $\text{ACO}_{\mathbb{B}}$. Third, $\text{ACO}_{\mathbb{B}}$ adapts faster than GA and PBIL. Hence, better overall performance is achieved during the dynamic changes.

For future work, $\text{ACO}_{\mathbb{B}}$ can be applied to the dynamic knapsack problem, which is closer to a real-world application. In fact, the performance of $\text{ACO}_{\mathbb{B}}$ can be furthermore improved since heuristic information is available to the knapsack problem using the weights and profits [6].

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant EP/K001310/1.

References

1. Alaya, I., Solnon, C., Ghédira, K.: Ant algorithm for the multi-dimensional knapsack problem. In: International Conference on Bioinspired Optimization Methods and their Applications. pp. 63–72 (2004)
2. Angus, D., Hendtlass, T.: Ant colony optimization applied to dynamically changing problem. In: Developments in Applied Artificial Intelligence. LNAI, vol. 2358, pp. 618–627. Springer-Verlag (2002)

3. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Proceedings of the European Conference on Artificial Life. pp. 134–142. Elsevier (1991)
4. Dorigo, M., Stützle, T. (eds.): Ant colony optimization. MIT Press, London, England (2004)
5. Fernandes, C.M., Rosa, A.C., Ramos, V.: Binary ant algorithm. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. pp. 41–48. GECCO '07, ACM, New York, NY, USA (2007)
6. Fidanova, S.: Aco algorithm for mkp using various heuristic information. In: Dimov, I., Lirkov, I., Margenov, S., Zlatev, Z. (eds.) Numerical Methods and Applications, Lecture Notes in Computer Science, vol. 2542, pp. 438–444. Springer Berlin Heidelberg (2003)
7. Guntsch, M., Middendorf, M., Schmeck, H.: An ant colony optimization approach to dynamic tsp. In: Proceedings of the 2001 Genetic and Evolutionary Computation Conference. pp. 860–867 (2001)
8. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments - a survey. IEEE Transactions on Evolutionary Computation 9(3), 303–317 (2005)
9. Ke, L., Feng, Z., Ren, Z., Wei, X.: An ant colony optimization approach for the multidimensional knapsack problem. Journal of Heuristics 16(1), 65–83 (2010)
10. Kong, M., Tian, P.: Introducing a binary ant colony optimization. In: Dorigo, M., Gambardella, L., Birattari, M., Martinoli, A., Poli, R., Sttzle, T. (eds.) Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science, vol. 4150, pp. 444–451. Springer Berlin Heidelberg (2006)
11. Kong, M., Tian, P., Kao, Y.: A new ant colony optimization algorithm for the multidimensional knapsack problem. Computers & Operations Research 35(8), 2672 – 2683 (2008)
12. Leguizamon, G., Michalewicz, Z.: A new version of ant system for subset problems. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. vol. 2, pp. 1459–1464 (1999)
13. Mavrovouniotis, M., Yang, S.: Ant colony optimization with immigrants schemes for the dynamic vehicle routing problem. In: EvoApplications2012: Applications of Evolutionary Computation. LNCS, vol. 7248, pp. 519–528. Springer-Verlag (2012)
14. Mavrovouniotis, M., Yang, S.: Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors. Applied Soft Computing 13(10), 4023–4037 (2013)
15. Montemanni, R., Gambardella, L.M., Rizzoli, A.E., Donati, A.V.: Ant colony system for a dynamic vehicle routing problem. Combinatorial Optimization 10, 327–343 (2005)
16. Nguyen, T.T., Yang, S., Branke, J.: Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation 6, 1–24 (2012)
17. Stützle, T., Hoos, H.: The MAX-MIN ant system and local search for the traveling salesman problem. In: Proceedings of the 1997 IEEE International Conference on Evolutionary Computation. pp. 309–314. IEEE Press (1997)
18. Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proceedings of the 2003 IEEE Congress on Evolutionary Computation. pp. 2246–2253. IEEE Press (2003)
19. Yang, S.: Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. Evolutionary Computation 16(3), 385–416 (2008)
20. Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. IEEE Trans. Evol. Comput. 12(5), 542–561 (2008)